Engineering 50 (2025) 117–127

Contents lists available at ScienceDirect

Engineering

journal homepage: www.elsevier.com/locate/eng

Research
Intelligent Manufacturing—Article

An Effective Local Search Algorithm for Flexible Job Shop Scheduling in Intelligent Manufacturing Systems

Junjie Zhang [a], Zhipeng Lü [a,b], Junwen Ding [a], Zhouxing Su [a], Xinyu Li [b], Liang Gao [b,*]

[a] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[b] *The State Key Laboratory of Intelligent Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

ARTICLE INFO

Article history:
Received 6 July 2023
Revised 28 June 2024
Accepted 3 July 2024
Available online 29 August 2024

Keywords:
Job shop scheduling
Adaptive weighting technique
Intelligent manufacturing systems

ABSTRACT

As one of the most classical scheduling problems, flexible job shop scheduling problems (FJSP) find widespread applications in modern intelligent manufacturing systems. However, the majority of meta-heuristic methods for solving FJSP in the literature are population-based evolutionary algorithms, which are complex and time-consuming. In this paper, we propose a fast effective single-solution based local search algorithm with an innovative adaptive weighting-based local search (AWLS) technique for solving FJSP. The adaptive weighting technique assigns weights to each operation and adaptively updates them during the exploration. AWLS integrates a Tabu Search strategy and the adaptive weighting technique to smooth the landscape of the search space and enhance the exploration diversity. Computational experiments on 313 well-known benchmark instances demonstrate that AWLS is highly competitive with state-of-the-art algorithms in terms of both solution quality and computational efficiency, despite of its simplicity. Specifically, AWLS improves the previous best-known results in the literature on 33 instances and match the best-known results on the remaining ones except for only one under the same time limit of up to 300 s. As a strongly non-deterministic polynomia (NP)-hard problem which has been extensively studied for nearly half a century, breaking the records on these classic instances is an arduous task. Nevertheless, AWLS establishes new records on 8 challenging instances whose previous best records were established by a state-of-the-art meta-heuristic algorithm and a famous industrial solver.

1. Introduction

Given the ongoing advancements in manufacturing and information technology, the landscape of production is transforming toward a paradigm characterized by "multi-variety, small batch production, shorter production cycles, and minimized work-in-progress inventory" [1,2]. Traditional manufacturing systems and control methodologies are increasingly unable to meet the demands of the new era. With the introduction of initiatives such as "Industry 4.0" in Germany and "Made in China 2025" in China, a new wave of industrial revolution and transformation is emerging [3]. Enhancing manufacturing systems necessitates improvements across various domains including flexibility, informatization, digitalization, and intelligence [4]. Within such manufacturing system, intelligent manufacturing has emerged as a critical technology driving the next-generation industrial revolution.

The job shop scheduling problem (JSP) is a focal point in modern intelligent manufacturing systems [5]. It represents a fundamental scheduling challenge in operations management, involving the sequencing across a defined set of machines to minimize the total completion time or makespan. As an extension of job shop scheduling, flexible job shop scheduling problem (FJSP) is an even more challenging non-deterministic polynomia (NP)-hard problem with numerous applications in intelligent manufacturing [6]. Unlike JSP, FJSP assigns a specific set of candidate machines to each operation, where processing times may vary across these machines. The optimization goal of FJSP aims to enhance operational efficiency and productivity within manufacturing systems by minimizing the overall job completion time.

As demonstrated in Ref. [7], population-based evolutionary algorithms have demonstrated effectiveness in generating high-quality solutions for FJSP. However, a significant challenge faced

---

* Corresponding author.
E-mail address: gaoliang@mail.hust.edu.cn (L. Gao).

https://doi.org/10.1016/j.eng.2024.07.022

by these algorithms is the management of a large population and the preservation of diversity throughout the search process, which can become excessively time-consuming. Addressing this issue often requires innovative strategies to balance exploration and exploitation efficiently while optimizing the scheduling problem efficiently [8]. Therefore, a simple single-solution based local search algorithm is proposed, which is rarely studied in current literature. Single-solution based algorithms are more efficient than population-based evolutionary algorithms since they do not need to manage a large population or involve crossover and mutation operations of individuals in the population. However, the search diversity of single-solution based algorithms is usually inferior to that of population-based evolutionary algorithms. To tackle this problem, we introduced a novel adaptive weighting technique, which can avoid repeatedly searching in the same search space as much as possible and improve the search diversity.

The following sections of this paper are structured as follows: Section 2 reviews the related literature, and Section 3 describes the proposed innovative adaptive weighting-based local search (AWLS) technique algorithm. Section 4 offers a comparative analysis of AWLS with state-of-the-art algorithms, while Section 5 discusses the advantages of the key components of our proposed AWLS algorithm. Section 6 concludes the paper.

2. Literature review

There exist two primary categories of algorithms widely used to tackle FJSP: exact algorithms and meta-heuristic algorithms. Exact algorithms rely on mathematical modeling methodologies, such as Lagrangian relaxation and mixed integer linear programming (ILP). Meta-heuristic algorithms include heuristic and meta-heuristic strategies such as rule-based heuristic, local search method, genetic algorithms, and so on. These algorithms provide approximate solutions through effective exploration of the search space. Additionally, artificial intelligence (AI) techniques are also applied to manage complex production scheduling hurdles, including JSP and FJSP [9,10]. However, AI-based algorithms are still in their infancy, while exact and meta-heuristic algorithms are still the primary focus due to their maturity and effectiveness in addressing such complex scheduling problems.

Most approaches for tackling FJSP using exact algorithms are based on ILP. Sawik [11] devised a multi-level integer linear program for modeling classical FJSP. Gomes et al. [12] introduced an innovative ILP model tailored to discrete parts manufacturing industries. The model considered factors such as parallel machines, limited buffers, and minimal setup effects, effectively addressing issues in scheduling literature. By utilizing commercial MILP software (International Business Machines Corporation, USA), the model achieved high-quality solutions for realistic scenarios, showcasing the competitiveness of this approach in solving JSP. Elazeem et al. [13] proposed a dual problem of the primal problem to solve FJSP, suggesting a quality measurement based on their differences. Moreover, Gomes et al. [14] introduced a novel mixed integer linear program model specifically for multi-objective FJSP. Different from typical FJSP models, this approach accounted for scenarios where a job could re-enter the same machine. Yin et al. [15] elucidated a novel mathematical model, demonstrating energy efficiency and environmental impact reduction in workshop operations. Kaplanoğlu [16] proposed an innovative algorithm based on an objective–oriented approach, streamlining the concise representation of FJSP. Specifically, by employing objective-oriented design, the solutions for FJSP could be encoded using a single encoding scheme rather than the complex two-string scheme commonly found in existing literature. Nevertheless, the computational challenges posed by NP-hardness usually lead to unsatisfactory performance of exact algorithms when applied to large-scale production scheduling problems [17].

Most meta-heuristic algorithms for solving FJSP involve the use of population-based algorithms. Pezzella et al. [18] laid out an innovative initialization approach and a crossover operator to enhance population diversity. Gao et al. [19] simultaneously optimized two operations to further refine the local optima of moving a single operation. Their genetic algorithm discovered 38 new better solutions. Zhang et al. [20] integrated Particle Swarm Optimization (PSO) and Tabu Search (TS) strategy to tackle FJSP. Local search efficiently identifies high-quality local optimal solutions of better quality, while global search prevents becoming trapped in local optima. PSO integrated a local search scheme with global search to enhance the search efficiency. Zhou et al. [21] developed an effective decoding strategy and a hybrid initialization method for addressing FJSP. They introduced a novel population updating strategy to maintain a balance between intensification and diversification in the search process. González et al. [22] affirmed a new neighborhood structure and a dissimilarity measure tailored to address FJSP effectively. Through experimental validation on benchmark instances for FJSP, their proposed algorithm outperformed existing methods. Palacios et al. [23] introduced a novel approach, combining genetic algorithms and heuristic seeding to solve fuzzy FJSP. Their algorithm substantiated competitive performance by integrating genetic algorithm with TS strategy. Li and Gao [24] proposed a hybrid algorithm that amalgamates the global exploration capability of genetic algorithm and the local exploitation capability of TS. This ingenious approach achieved outstanding performance across various benchmark instances through strategic encoding methods, genetic operators, and neighborhood structures. Kemmoé-Tchomté et al. [25] advanced an enhanced Greedy Randomized Adaptive Search Procedure (GRASP) by incorporating diversification by GRASP and intensification by multi-level evolutionary local search. Their algorithm can obtain the best-known solutions on 125 benchmark instances. Caldeira and Gnanavelbabu [26] merged improved initialization, local search, and acceptance criteria to overcome local optima and enhance solution quality.

In recent times, spurred by rapid advancement in computer technology and theoretical developments in the field, several competitive algorithms have emerged to address FJSP. Ding et al. [8] used a novel approach to calculate solution distances, integrating it with a path relinking strategy as a way to enhance the solution quality significantly. Their algorithm broke the world records for 10 benchmark instances. Fan et al. [27] introduced an innovative hybrid algorithm that combined the Jaya method with TS, demonstrating exceptional stability and solution quality across a range of benchmark data sets by implementing unique Jaya operators and customized local search strategies. Li et al. [28] advanced a highly efficient two-stage evolutionary algorithm driven by domain-specific heuristics for initial population generation and Pareto-based techniques for solution convergence. Experimentation on a benchmark data set with 20 instances validated the effectiveness of their algorithm. Zhang et al. [9] innovated with a deep reinforcement learning network-based method to tackle dynamic FJSP, introducing an adaptive learning technique to enhance decision-making under varying conditions. Du et al. [10] developed a deep Q-learning network model aimed at optimizing makespan and energy consumption simultaneously through a weighted approach, incorporating specific features like crane transportation stages to boost performance. Xie et al. [29] combined global search capability from genetic algorithms with local search power from TS to improve search efficiency. They integrated a new neighborhood structure for JSP into TS for searching larger space of neighborhood solutions. Their algorithm found 13 new upper bounds on 69 distributed FJSP benchmark instances. Sun et al. [30] enhanced a hybrid genetic algorithm by prioritizing machine workload balance, introducing innovative strategies in chromosome representation, crossover, mutation operators, and local search techniques to address FJSP challenges effectively. Yang et al. [31] improved the dragonfly algorithm by incorporating a dynamic opposite learning strategy, achieving high-quality solutions on large-scale FJSP instances generated by the Brandimate rule. Alzaqebah et al. [32] introduced the brain storming optimization algorithm, enhancing global search capabilities through novel selection methods and neighborhood structures.

3. Adaptive weighting-based local search

To tackle FJSP, we propose an AWLS algorithm. This approach incorporates a TS strategy to prevent the algorithm from repeating recently visited solutions or attributes, as well as employing adaptive weighting techniques to smooth the landscape of the search space.

The primary framework of AWLS is outlined in Algorithm 1. First, AWLS generates an initial feasible solution randomly (line 1). This initialization method ensures a fair and unbiased allocation of operations to candidate machines. The current solution is denoted as S. Then, the best found solution and the last solution are set as the initial solution and weights are initialized (lines 2 and 3). Next, AWLS performs the moves selected from neighborhood moves to improve the incumbent solution (lines 4–16).

Algorithm 1. The main framework of AWLS

```
Input: FJSP instance
Output: the best found solution $S^*$
1:  $S^* \leftarrow S \leftarrow$ init()
2:  $f^* \leftarrow f' \leftarrow S$.makespan
3:  Tabu list TL $\leftarrow \varnothing$, $w_i \leftarrow 0$, $t_i \leftarrow +\infty$, $i \in S$
4:  while stopping condition is not reached do
5:      $(o^*, m, k) \leftarrow$ FindMove($S$, TL) /* Algorithm 2 */
6:      $S \leftarrow S \oplus (o^*, m, k)$
7:      TL. append($(o^*, m, k)$)
8:      $f \leftarrow S$. makespan
9:      cp $\leftarrow S$. criticalpath
10:     UpdateWeight($o^*$, cp, $f^*$, $f'$, $f$) /* Algorithm 3 */
11:     $f' \leftarrow f$
12:     if $f^* > f$ then
13:         $S^* \leftarrow S$
14:         $f^* \leftarrow f$
15:     end if
16: end while
17: Return $S^*$
```

f^*: best makespan; f': last makespan; cp: critical path; f: current makespan; o^*: moved operation; ϕ: NULL.

In particular, the proposed method for evaluating neighborhoods is utilized to estimate the potential moves of the incumbent solution. Subsequently, a promising move is chosen based on the estimated neighborhood values and their status in the Tabu list (line 5). After that, a new incumbent solution is obtained by performing the promising move (lines 6 and 7), while the makespan and critical path of the new incumbent solution are calculated (lines 8 and 9). The move (o^*, m, k) indicates that operation o^* is moved to position k on machine m. Then, the weights of the corresponding operations are updated based on the makespan changes after the move is performed (line 10). At last, the best found solution S^* will be updated when S is better than S^* (lines 12–14).

3.1. Neighborhood evaluation and adaptive weighting technique

A solution of FJSP is represented as (α, π) as used in Ref. [22], where α is a feasible assignment of each operation o to a machine and π denotes the sequence of operations on each machine. In our AWLS, a feasible assignment of operation o on machine m at position k is denoted as (o, m, k). Algorithm 2 describes the detailed neighborhood move selection procedure. The critical path cp is first obtained based on the current solution S followed by the generation of candidate neighborhood moves based on two distinct neighborhoods. In this paper, the k-insertion neighborhood (N^k) proposed by Ref. [33] is employed to generate candidate machine re-assignments, while the N^7 neighborhood proposed by Ref. [20] is employed to generate candidate position changes on the same machine. Next, AWLS attempts to reassign the critical operations to other feasible positions on the same machine m_u (line 5). Procedure PosiEstimate() is utilized to estimate the non-tabued feasible candidate position changes (u, m_u, j), and the position change with the minimum estimated makespan for all feasible candidate positions is selected (lines 5–11). Similarly, machine re-assignment with the minimum estimated makespan for all feasible positions on all candidate machines is selected (lines 12–18). Finally, the move with the best evaluated makespan is selected and returned (line 20).

Algorithm 2: FindMove(S, TL)

```
Input: solution $S$, Tabu list TL
Output: the selected move $(o^*, m, k)$
1:  Selected move $u$, $m'$, $j'$ $\leftarrow$NULL
2:  Estimated makespan of the selected move $\bar{f} \leftarrow +\infty$
3:  cp$\leftarrow S$.criticalpath
4:  for all operation $u$ in cp do
5:      for all feasible position $j$ on machine $m_u$ do
6:          $\bar{f}' \leftarrow$ PosiEstimate($(u, m_u, j)$), $(u, m_u, j) \in N^7$, $(u, m_u, j) \notin$ TL
7:          if $\bar{f} > \bar{f}'$ then
8:              $(o^*, m, k) \leftarrow (u, m_u, j)$
9:              $\bar{f} \leftarrow \bar{f}'$
10:         end if
11:     end for
12:     for all position $j'$ on each candidate machine $m'$ do
13:         $\bar{f}' \leftarrow$ MachEstimate($(u, m', j')$), $(u, m', j') \in N^k$, $(u, m', j') \notin$ TL
14:         if $\bar{f} > \bar{f}'$ then
15:             $(o^*, m, k) \leftarrow (u, m', j')$
16:             $\bar{f} \leftarrow \bar{f}'$
17:         end if
18:     end for
19: end for
20: Return $(o^*, m, k)$
```

In classical local search, neighborhood move with the minimum estimated makespan is selected. However, this approach has two drawbacks. First, the method for estimating makespan is inaccurate, so the move with the minimum estimated makespan may not achieve the expected results (estimated makespan). Second, greedily selecting the neighborhood move with the minimum estimated makespan can easily lead the search to fall into a local optimum trap.

To tackle these two issues, we introduce an adaptive weighting strategy to modify the traditional neighborhood evaluation. First of all, the weights of all the operations are initialized to zero at the beginning of the algorithm. The weight of the moved operation is increased while the incumbent solution cannot be improved by performing a move, which leads to a larger makespan in the following evaluation when involving this operation. In this way, the algorithm will avoid choosing neighborhood moves based on the modified neighborhood evaluation involving this operation. Furthermore, when the search falls into a local optimum, weighting the operations involved in moves that have not improved the current solution can smooth the landscape of the search space and improve search efficiency.

Suppose u and v are the two different operations, and u precedes v as shown in Fig. 1, after moving u to the rear of v, the new makespan is estimated as:

$$\text{makespan}^{u,v} = \max\left\{R^{u,v}[i] + p[i] + Q^{u,v}[i] + Z(i)\right\}, \forall i \in \left\{L_1, \cdots, L_g, v, u\right\} \tag{1}$$

where $p[i]$ denotes the processing time of operation i. $R^{u,v}$ and $Q^{u,v}$ are calculated using the method as introduced in Ref. [22]. L_g represents the gth operation after operation u, In Eq. (1), $Z(i)$ is regarded as the adaptive additional processing time, which is a certain proportion of its cumulative weight and is defined as Eq. (2).

$$Z(i) = \max\left(\left(1 - \frac{t_i}{\text{rand}(1,\gamma)}\right) \times w_i, 0\right) \tag{2}$$

where w_i represents the cumulative weight of operation i, and t_i represents the idle count of operation i, which denotes the most recent local search iteration when operation i becomes a critical operation. t_i and w_i are respectively initialized to $+\infty$ and 0 and are updated at each iteration (Section 3.2). Specifically, the idle count of operation t_i and parameter γ jointly determine the proportion of the cumulative weight in calculating the additional processing time $Z(i)$ of operation i. In Eq. (2), $\text{rand}(1,\gamma)$ denotes a random integer between 1 and γ to improve the randomness of the algorithm and improve search diversity. Similarly, the makespan estimation of the machine re-assignment can be calculated in the same way.

For the insertion neighborhood move illustrated in Fig. 1, the $R^{u,v}$ and $Q^{u,v}$ estimated values of the operations in the processing sequence are calculated as follows

$$R^{u,v}[L_1] = \max\{R[\text{JP}[L_1]] + p[\text{JP}[L_1]], R[\text{MP}[u]] + p[\text{MP}[u]]\} \tag{3}$$

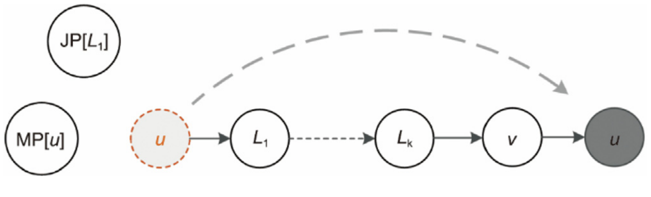

Fig. 1. Illustration of insertion move.

where JP[i] and MP[i] represent the job predecessor and machine predecessor of operation i , respectively. $R[i]$ and $Q[i]$ are the longest path from the starting node to the operation i and the longest path from the operation i to the ending node, respectively.

For other operation $i \in \{L_2, \ldots, L_g, v\}$:

$$R^{u,v}[i] = \max\{R[\text{JP}[i]] + p[\text{JP}[i]], R^{u,v}[\text{MP}[i]] + p[\text{MP}[i]]\} \tag{4}$$

For operation u:

$$R^{u,v}[u] = \max\{R[\text{JP}[u]] + p[\text{JP}[u]], R^{u,v}[v] + p[v]\} \tag{5}$$

The corresponding $Q^{u,v}$ values are calculated as follows.

$$Q^{u,v}[u] = \max\{Q[\text{JS}[u]] + p[\text{JS}[u]], Q[\text{MS}[v]] + p[\text{MS}[v]]\} \tag{6}$$

where JS[i] and MS[i] denote the job successor and machine successor of operation i, respectively.

$$Q^{u,v}[v] = \max\{Q[\text{JS}[v]] + p[\text{JS}[v]],\ Q^{u,v}[u] + p[u]\} \tag{7}$$

For other operations $i \in \{L_g, \ldots, L_1\}$

$$Q^{u,v}[i] = \max\{Q[\text{JS}[i]] + p[\text{JS}[i]],\ Q^{u,v}[\text{MS}[i]] + p[\text{MS}[i]]\} \tag{8}$$

AWLS performs the best move from the non-tabued candidate moves, and refrains from executing the same move within the Tabu tenure [34]. However, too long Tabu tenure will lead to heavy computational time and may miss promising solutions.

In AWLS, w_i records the cumulative weight of operation i in the search, which is used to prevent local search from returning to previously visited solutions. However, using the cumulative weight directly as additional processing time for each operation in estimating the neighborhood move will greatly change the landscape of the search space. Therefore, the idle count t_i is used to adaptively adjust the impact of the cumulative weight on the neighborhood evaluation.

From a macroscopic perspective, the increment in the idle count t_i of operation i indicates that operation i has not been on the critical path for a long period, which indicates that there could be a notable distinction between the current solution and those previously visited ones. Therefore, the cumulative weight of operation i should have little impact on the neighborhood estimation.

In addition, TS can prevent the search from repeating recently visited solutions or attributes by utilizing the Tabu list and Tabu tenure. On the other hand, the adaptive weighting technology can aid the search in escaping local optima trap by adaptively adjusting the objective function value in a smoother manner. Thus, it can be considered as a long-term memory strategy to prevent the search from repeating visited solutions or attributes. In general, these two mechanisms are complementary with each other in terms of jumping out of local optimum trap. Therefore, our AWLS integrates TS and adaptive weighting technology to enhance the diversification of the search.

3.2. Weight updating procedure

It is well known that the weight updating is one of the most central components of weighting-based algorithms. In the proposed weight updating procedure (Algorithm 3), AWLS adaptively adjusts the landscape of the search by changing the values of the cumulative weight and idle count, with the aim to smooth the landscape of the search space. In details, the weight updating procedure is presented in lines 1–7, while how the idle count is updated is described in lines 8–21. The weight and idle count resetting are given in lines 22–27.

Algorithm 3: UpdateWeight$(o^*, \text{cp}, f^*, f', f)$.

```
Parameter: $o^*$, cp, $f^*$, $f'$, $f$
/*lines 1–7: Weight updating */
1:  if $f \geq f'$ then
2:      if $t_{o*} > \beta$ then
3:          $w_{o*} \leftarrow 0$                 /* Forget cumulative weight */
4:      else
5:          $w_{o*} \leftarrow w_{o*} + 1$        /* Increase weight by one */
6:      end if
7:  end if
    /* lines 8–21: Idle count updating */
8:  if $f \geq f'$ then
9:      if $t_{o*} > \gamma$ then
10:         $t_{o*} \leftarrow \gamma$            /* Decrease $t_{o^*}$ to $\gamma$ */
11:     else
12:         $t_{o*} \leftarrow \max\{t_{o*} - \theta, 0\}$   /* Decrease $t_{o^*}$ by $\theta$ */
13:     end if
14:     for all operation $o \in S\backslash \text{cp}\backslash o^*$ do
15:         $t_o \leftarrow t_o + 1$              /* Increase $t_o$ by one for other operations */
16:     end for
17: else
18:     for all operation $o \in S$ do
19:         $t_o \leftarrow t_o + 1$              /* Increase $t_o$ by one for all operations */
20:     end for
21: end if
    /* lines 22–27: Weight and idle count resetting */
22: if $f < f^*$ then
23:     for all operation $o \in S$ do
24:         $t_o \leftarrow +\infty$
25:         $w_o \leftarrow 0$
26:     end for
27: end if
```

t_{o*}, w_{o*}, w_o, and t_o denotes the idle count of operation i, β, γ, θ are show in Table 1.

If the last solution does not improve after performing a move operation (lines 1–7), the weight of the moved operation o^* is reset to 0 when its idle count exceeds β (lines 2–3). This indicates that operation o^* has not been moved for a long period. Otherwise, its weight is increased by one (lines 4–5).

Then, the idle count of the corresponding operations will be updated. On one hand, the idle count of the moved operation is updated according to γ and θ when the incumbent solution is not better than the last one (lines 8–13). Similar to weight updating, the idle count of the moved operation is decreased to γ if its idle count exceeds γ (line 10). Otherwise, its idle count is decreased by θ each time (line 12). Then, the idle counts of operations other than the critical operations and the moved operation increase by one (lines 14–15), while the idle count of the critical operations remains unchanged. In the latter case, it maintains the impact of the cumulative weight on the search and prevents the algorithm from repeating recently visited solutions or attributes. On the other hand, when the incumbent solution is better than the last one, the idle count of all operations increases by one, which reduces the impact of the weight on the search (lines 18–20).

When a new best solution is obtained, the weight and idle count of all operations are reset to $+\infty$ and 0 (lines 22–27), respectively. In this case, $Z(o)$ is equal to zero and the estimated makespan becomes the original estimated value as used in previous studies. In other words, the weighting technique is deactivated. If a new best solution is obtained, it signifies that AWLS may explore a new region. Thus, the weights used in the previous search region may not be suitable for the new one, so the weights and idle counts are reset.

In general, when the search encounters stagnation, the cumulative weight of the moved operation is increased while the idle count of critical operations is decreased (lines 5 and 9–13), which can smooth the landscape of the search space and improve search efficiency. In contrast, when the makespan can be improved, the idle count of all operations increases and the impact of the cumulative weight of operations on the search decreases (lines 18–20). Finally, the accumulated weights and idle counts of all the operations are reset upon discovering a new best solution (lines 23–26).

4. Results and comparisons

4.1. Experimental instances and parameter settings

AWLS is implemented in C++ and executed on a Windows operating system running on an Intel Xeon E5-2698 processor (USA). Prior to conducting experiments, we used Geatpy (China), a tuning software, to optimize algorithm parameters. Geatpy conducted parameter tuning on a randomly selected group of 40 classic FJSP instances, setting the ranges for γ, β, and θ at [1, 200], [1, 5000], and [1, 50], respectively. Each parameter was initialized with a random value within the defined range during the tuning process, which employed the differential evolution method. Based on the results from Geatpy, we selected γ = 40, β = 500, and θ = 5 as the parameters demonstrating the best overall performance in our experiments (Table 1).

To evaluate the performance of AWLS, we conducted experiments on four famous benchmarks: DPdata [35], BCdata [36], BRdata [6], and HUdata [37], commonly referenced in the literature. For each instance, we executed 20 independent runs. The cutoff time limits on the BCdata, BRdata, DPdata, and HUdata benchmark are set to 90, 90, 300, and 300 s, respectively. Additionally, we provided comparison results with the master-apprentice evolutionary (MAE) algorithm, employing a cutoff time of one hour, which is consistent with the settings used in MAE.

AWLS is compared with the following state-of-the-art metaheuristics in the literature: scatter search with path relinking (SSPR) [22], hybrid genetic tabu search (HGTS) [23], hybrid genetic algorithm (HA) [24], multi-start multi-level evolutionary local search (GRASP-mELS) [25], improved Jaya algorithm (IJA) [26], MAE [8], and hybrid Jaya algorithm (HJ) [27], which are the best performing algorithms for solving FJSP to the best of our knowledge. Using the same approach as for MAE, computation time is standardized as computer-independent central processing unit time (CI–CPU). Specifically, the speed factor of HJ, MAE, IJA, GRASP-mELS, SSPR, HA, and HGTS are set to 1.07, 1.00, 0.63, 1.09, 0.75, 0.50, and 0.63, respectively, while the setting of our algorithm is set to 1. Following standard practice in the field, algorithms were compared using the following metrics: average relative percentage deviation (RPD), defined as RPD $= 100 \times (g - \text{LB})/\text{LB}$, where g represents the best found or average makespan in 20 runs and LB is

Table 1
Parameter settings of AWLS.

Parameter	Initial value range	Value	Description
γ	[1, 200]	40	Upper bound of random counts
β	[1, 5000]	500	Maximum rounds to retain cumulative weight
θ	[1, 50]	5	Rate of idle counts reduction

the lower bound. Additionally, the average running time (*t*) was calculated based on 20 independent runs.

4.2. Comparison with meta-heuristics

The comparison of our algorithm with the reference algorithms (SSPR, HGTS, HA, GRASP-mELS, IJA, MAE, and HJ) across the 313 benchmark instances is presented in Tables 2–5. The column "Ins." indicates the instances, avg indicates the average, UB indicates upper bound, and LB marked with * indicates the makespan of the optimal solutions.

The comprehensive comparative analysis conducted on the BCdata benchmark alongside reference meta-heuristics is elucidated in Table 2. AWLS records an average RPD of 0.06 and a running time of 17.71, both of which are smaller than those for SSPR, GRASP-mELS, and MAE. Additionally, AWLS's best and average RPD values are better than those of HGTS and HA, highlighting that AWLS outperforms HGTS and HA on 7 and 5 instances, respectively. The best RPD value of HJ is equal to that of AWLS, while it requires more computational time than AWLS. Furthermore, AWLS obtains the lower bounds on all the instances in this set.

In Table 3, AWLS has an average RPD of 0.58 and a CI–CPU of 6.89, surpassing all other reference algorithms. This observation signifies that AWLS can obtain superior solutions in a shorter time compared to other reference algorithms. Besides, AWLS obtains smaller best RPD than HA though AWLS requires slightly more computational time. As for MAE, its computational time is almost twice that of AWLS while its average RPD value is larger than that of AWLS.

In Table 4, AWLS shows the best and average RPD values of AWLS are 0.94 and 1.12, respectively, both smaller than those of SSPR and HGTS. When the algorithm achieves a smaller RPD, it indicates that it can obtain better solutions. AWLS can obtain superior solutions compared to SSPR and HGTS. Although AWLS may require slightly more computational time compared to HA,

Table 2
Results on the BCdata benchmark instances.

Ins.	LB	2015		2015		2016		2017		2019		2021		This paper	
		SSPR		HGTS		HA		GRASP-mELS		MAE		HJ		AWLS	
		best(avg)	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)
mt10c1	927*	**927**(928)	26	**927**(927)	13	**927**	12	**927**(927)	8	**927**(927.30)	45.72	**927**	43	**927**(927.30)	4.99
mt10cc	908*	**908**(908)	20	**908**(910)	13	**908**	10	**908**(909)	17	**908**(909.85)	14.25	**908**	36	**908**(908.40)	23.04
mt10x	918*	**918**(918)	23	**918**(918)	15	**918**	11	**918**(918)	2	**918**(918.00)	25.67	**918**	16	**918**(918.00)	4.94
mt10xx	918*	**918**(918)	19	**918**(918)	12	**918**	11	**918**(918)	2	**918**(918.00)	4.50	**918**	2	**918**(918.00)	3.06
mt10xxx	918*	**918**(918)	20	**918**(918)	12	**918**	11	**918**(918)	2	**918**(918.00)	6.78	**918**	9	**918**(918.00)	2.45
mt10xy	905*	**905**(906)	21	**905**(905)	13	**905**	11	**905**(905)	26	**905**(905.00)	34.42	**905**	16	**905**(905.10)	29.98
mt10xyz	847*	**847**(847)	20	**847**(850)	18	**847**	9	**847**(847)	26	**847**(847.65)	35.46	**847**	36	**847**(847.00)	21.09
setb4c9	914*	**914**(916)	28	**914**(914)	16	**914**	15	**914**(914)	11	**914**(918.25)	39.78	**914**	60	**914**(914.00)	15.41
setb4cc	907*	**907**(907)	21	**907**(908)	15	**907**	15	**907**(907)	29	**907**(907.00)	12.54	**907**	33	**907**(907.00)	6.66
setb4x	925*	**925**(925)	19	**925**(925)	15	**925**	13	**925**(925)	4	**925**(925.00)	16.42	**925**	14	**925**(925.00)	3.42
setb4xx	925*	**925**(925)	21	**925**(925)	14	**925**	5	**925**(925)	2	**925**(925.00)	7.70	**925**	22	**925**(925.00)	2.91
setb4xxx	925*	**925**(925)	22	**925**(925)	15	**925**	9	**925**(925)	3	**925**(925.00)	8.45	**925**	13	**925**(925.00)	3.21
setb4xy	910*	**910**(912)	32	**910**(910)	19	**910**	12	**910**(910)	18	**910**(910.00)	58.79	**910**	27	**910**(910.00)	9.78
setb4xyz	902*	905(905)	21	905(905)	15	905	14	**902**(904)	11	**902**(905.60)	34.60	**902**	14	**902**(904.50)	14.76
seti5c12	1169*	1170(1173)	25	1170(1171)	41	1170	31	**1169**(1172)	39	1170(1174.40)	64.13	**1169**	144	**1169**(1170.05)	23.98
seti5cc	1135*	**1135**(1136)	29	1136(1137)	34	**1136**	17	**1135**(1136)	24	**1135**(1136.20)	32.41	**1135**	90	**1135**(1135.50)	39.69
seti5x	1198*	**1198**(1199)	41	1199(1201)	38	**1198**	27	**1198**(1199)	36	**1198**(1201.60)	75.48	**1198**	55	**1198**(1198.90)	24.56
seti5xx	1194*	1197(1199)	37	1197(1198)	34	1197	29	**1194**(1197)	26	1197(1198.50)	45.76	**1194**	606	**1194**(1198.25)	17.61
seti5xxx	1194*	**1194**(1198)	38	1197(1198)	31	1197	19	**1194**(1197)	27	1197(1198.45)	35.50	**1194**	38	**1194**(1197.95)	33.35
seti5xy	1135*	**1135**(1136)	29	1136(1137)	34	1136	17	**1135**(1136)	28	**1135**(1136.40)	25.53	**1135**	60	**1135**(1135.30)	45.15
seti5xyz	1125*	**1125**(1126)	35	**1125**(1126)	43	**1125**	33	**1125**(1127)	42	**1125**(1128.75)	32.96	**1125**	166	**1125**(1125.35)	41.46
RPD	—	0.03(0.12)	—	0.07(0.13)	—	0.05	—	0(0.07)	—	0.03(0.17)	—	0	—	0(0.06)	—
CI—CPU	—	19.54	—	13.8	—	7.88	—	19.88	—	31.28	—	76.43	—	17.71	—
#better	—	3	—	7	—	6	—	0	—	3	—	0	—	—	—
#even	—	18	—	14	—	15	—	21	—	18	—	21	—	—	—
#worse	—	0	—	0	—	0	—	0	—	0	—	0	—	—	—

Table 3
Results on the BRdata benchmark instances.

Ins.	LB	2015		2015		2016		2017		2019		2019		2021		This paper	
		SSPR		HGTS		HA		GRASP-mELS		IJA		MAE		HJ		AWLS	
		best(avg)	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)
Mk01	40*	**40**(40)	11	**40**(40)	5	**40**	0	**40**(40)	0	**40**	0.15	**40**(40.00)	0.20	**40**	0	**40**(40.00)	0
Mk02	26*	**26**(26)	15	**26**(26)	15	**26**	1	**26**(26)	10	**27**	2.41	**26**(26.00)	0.55	**26**	1	**26**(26.00)	0.33
Mk03	204*	**204**(204)	24	**204**(204)	2	**204**	0	**204**(204)	0	**204**	1.19	**204**(204.00)	0.16	**204**	0	**204**(204.00)	0
Mk04	60*	**60**(60)	19	**60**(60)	10	**60**	0	**60**(60)	0	**60**	3.82	**60**(60.00)	0.47	**60**	1	**60**(60.00)	0.03
Mk05	172*	**172**(172)	57	**172**(172)	18	**172**	5	**172**(173)	15	**172**	3.08	**172**(172.00)	1.46	**172**	12	**172**(172.00)	0.31
Mk06	57*	**57**(58)	40	**57**(58)	63	**57**	54	**58**(58)	36	**57**	37.13	**57**(58.15)	30.40	**57**	159	**57**(57.95)	15.07
Mk07	139*	**139**(141)	84	**139**(139)	33	**139**	20	**139**(140)	32	**139**	25.56	**139**(139.70)	61.58	**139**	176	**139**(139.10)	23.65
Mk08	523*	**523**(523)	83	**523**(523)	3	**523**	0	**523**(523)	0	**523**	0.51	**523**(523.00)	0.36	**523**	0	**523**(523.00)	0
Mk09	307*	**307**(307)	52	**307**(307)	24	**307**	1	**307**(307)	0	**307**	19.83	**307**(307.00)	1.13	**307**	2	**307**(307.00)	0.06
Mk10	189	196(197)	94	198(199)	104	197	33	197(199)	59	197	60.62	195(195.95)	36.78	196	448	195(196.60)	29.39
RPD	—	0.37(0.74)	—	0.48(0.71)	—	0.42	—	0.6(0.83)	—	8.08	—	0.31(0.61)	—	0.37	—	0.31(0.58)	—
CI—CPU	—	35.93	—	17.45	—	5.7	—	16.57	—	9.72	—	13.31	—	85.49	—	6.89	—
#better	—	1	—	1	—	1	—	2	—	2	—	0	—	1	—	—	—
#even	—	9	—	9	—	9	—	8	—	8	—	10	—	9	—	—	—
#worse	—	0	—	0	—	0	—	0	—	0	—	0	—	0	—	—	—

Table 4
Results on the DPdata benchmark instances.

Ins.	LB	2015		2015		2016		2017		2019		2019		This paper	
		SSPR		HGTS		HA		GRASP-mELS		IJA		MAE		AWLS	
		best(avg)	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)	best	*t* (s)	best(avg)	*t* (s)	best(avg)	*t* (s)
01a	2505*	**2505**(2508)	68	**2505**(2505)	122	**2505**	108	**2505**(2505)	62	**2505**	97.1	**2505**(2505.00)	28.56	**2505**(2505.00)	21.36
02a	2228*	2229(2230)	100	2230(2234)	205	2230	133	2229(2231)	86	2230	198.0	**2228**(2230.70)	145.12	**2228**(2229.50)	131.45
03a	2228*	**2228**(2228)	110	**2228**(2230)	181	2229	97	**2228**(2230)	94	2230	61.6	**2228**(2228.00)	55.80	**2228**(2228.30)	107.21
04a	2503*	**2503**(2504)	57	**2503**(2503)	112	**2503**	87	**2503**(2503)	31	2503	107.2	**2503**(2503.00)	8.62	**2503**(2503.00)	10.78
05a	2192	2211(2215)	112	2214(2218)	208	2212	116	2212(2215)	126	2210	112.1	2208(2211.45)	125.69	**2207**(2210.30)	141.80
06a	2163	2183(2192)	181	2193(2198)	260	2197	93	2195(2200)	181	**2182**	94.2	**2182**(2188.85)	177.42	2185(2190.10)	167.05
07a	2216	2274(2285)	139	2270(2280)	344	2279	204	2276(2284)	127	2270	242.1	2269(2274.60)	180.30	**2250**(2261.20)	206.78
08a	2061*	2064(2066)	181	2070(2074)	318	2067	184	2069(2072)	144	2065	170.2	**2063**(2064.30)	122.58	**2063**(2064.30)	200.04
09a	2061*	2062(2063)	213	2067(2069)	376	2065	201	2069(2071)	170	2065	165.4	**2062**(2063.15)	176.44	**2062**(2062.70)	194.98
10a	2212	2269(2287)	120	2247(2266)	369	2287	238	2263(2278)	110	2252	286.4	2247(2266.40)	224.36	**2241**(2252.10)	175.27
11a	2018	2051(2058)	193	2064(2069)	294	2060	181	2065(2068)	170	2057	160.1	**2050**(2051.80)	200.57	**2050**(2051.60)	215.51
12a	1969	2018(2020)	280	2027(2033)	486	2027	151	2039(2045)	148	2020	170.2	**2016**(2021.45)	215.64	**2016**(2022.35)	233.46
13a	2197	2248(2257)	119	2250(2264)	416	2248	293	2252(2263)	158	2250	246.5	2247(2251.75)	116.55	**2235**(2239.15)	202.16
14a	2161*	2163(2164)	269	2170(2173)	396	2167	210	2170(2174)	191	2164	291.9	**2163**(2163.90)	191.26	**2163**(2163.75)	241.37
15a	2161*	2162(2163)	376	2168(2169)	523	2163	192	2172(2174)	173	2163	239.7	**2162**(2164.35)	203.20	**2162**(2163.80)	195.96
16a	2193	2244(2253)	131	2246(2257)	384	2249	160	2243(2258)	151	2250	205.6	2242(2251.65)	196.50	**2228**(2233.80)	226.56
17a	2088	2130(2134)	299	2142(2146)	483	2140	203	2145(2152)	190	2136	196.7	**2128**(2132.70)	245.71	**2128**(2134.25)	229.43
18a	2057	2119(2123)	409	2129(2133)	650	2132	133	2146(2151)	164	**2107**	162.4	2118(2124.85)	242.20	2118(2126.65)	234.79
RPD	–	1.18(1.4)	–	1.34(1.59)	–	1.43	–	1.49(1.73)	–	1.17	–	1.04(1.24)	–	0.94(1.12)	–
CI–CPU	–	170	–	214.45	–	82.89	–	149.94	–	112.21	–	158.70	–	174.22	–
#better	–	11	–	15	–	16	–	15	–	14	–	5	–	–	–
#even	–	6	–	3	–	2	–	3	–	2	–	12	–	–	–
#worse	–	1	–	0	–	0	–	0	–	2	–	1	–	–	–

Table 5
Results on the HUdata benchmark instances.

Ins.	edata GRASP-mELS		edata SSPR		edata MAE		edata AWLS		rdata GRASP-mELS		rdata SSPR		rdata MAE		rdata AWLS		vdata GRASP-mELS		vdata SSPR		vdata MAE		vdata AWLS	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
mt06-10-20	0	0	0	0.04	0	0.07	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
la01-05	0	0	0	0	0	0	0	0	0	0.07	0.07	0.09	0	0.07	0	0.02	0	0	0	0	0	0	0	0
la06-10	0	0	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0
la11-15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
la16-20	0	0	0	0	0	0	0	0	0	0.03	0	0	0	0	0	0	0	0	0	0	0	0	0	0
la21-25	0	0.24	0.08	0.23	0	0.22	0	0.06	2.63	3.27	2.53	2.91	1.91	2.35	1.91	2.30	0.49	0.80	0.23	0.35	0.10	0.24	0.10	0.24
la26-30	0.33	0.61	0.43	0.66	0.30	0.73	0.21	0.28	0.36	0.71	0.36	0.48	0.13	0.27	0.11	0.23	0.17	0.24	0.06	0.08	0	0.03	0	0.03
la31-35	0.05	0.11	0.01	0.07	0	0.01	0	0	0.05	0.12	0.04	0.05	0	0.02	0	0.02	0.04	0.07	0.01	0.02	0	0	0	0
la36-40	0	0.04	0	0.05	0	0.02	0	0	0.36	1.22	0.66	0.90	0	0.10	0	0.10	0	0	0	0	0	0	0	0
CI–CPU	22.98	–	28.26	–	18.56	–	17.66	–	51.23	–	34.78	–	44.68	–	44.58	–	30.25	–	47.83	–	25.76	–	17.96	–
#better	4.00	–	5.00	–	8.00	–	–	–	18.00	–	19.00	–	14.00	–	–	–	13.00	–	8.00	–	3.00	–	–	–
#even	39.00	–	38.00	–	35.00	–	–	–	25.00	–	24.00	–	29.00	–	–	–	30.00	–	35.00	–	40.00	–	–	–
#worse	0	–	0	–	0	–	–	–	0	–	0	–	0	–	–	–	0	–	0	–	0	–	–	–

GRASP-mELS, IJA, and MAE, it obtains the smallest best and average RPD values (0.94 and 1.12). Moreover, AWLS can establish new world record on instance 16a from 2231 to 2228 in 300 s.

From Table 5, it is evident that AWLS achieves smaller best and average RPD values across all the instances compared to other reference algorithms, while AWLS requires less computational time. In particular, AWLS breaks the previous world records established by GRASP-mELS, SSPR, and MAE on the 4–5–8, 18–19–14, and 13–8–3 instances of the edata, rdata, and vdata benchmark sets, respectively. In summary, under the time limit of up to 300 s, AWLS improves the best results obtained by SSPR, GRASP-mELS, and MAE for 47, 52, and 33 out of the 178 benchmark instances, respectively.

4.3. Comparison with the exact method

AWLS is compared with the state-of-the-art exact algorithm, Large Neighborhood Search (LNS) + Failure-directed Search (FDS) (International Business Machines Corporation, USA) [38], which relies on constraint programming. LNS + FDS serves as the core of the automated search mechanism within Constraint Programming Optimizer (CPO). Notably, CPO has undergone successful evaluations across diverse scheduling benchmarks such as JSP and FJSP.

The cutoff time for CPO is set to 8 h, while the cutoff time for AWLS is 1 h. AWLS outperforms CPO on 46 instances, matches CPO on 263 instances, and is outperformed by CPO on 4 instances out of the totally 313 instances. The results for the 46 improved instances where AWLS showed improvement are reported in Table 6.

4.4. Comparison with the commercial solver

The famous commercial solver Quintiq has obtained new world records for 119 instances [8]. However, it is important to note that Quintiq did not disclose any details about their algorithms or the time limits for achieving these results.

In our comparison between AWLS and Quintiq, the experimental results indicate that MAE outperforms Quintiq on 14 instances, matches results on 93 instances, and is outperformed by Quintiq on 14 instances out of 121 ones. AWLS obtains 8 new world records, which are reported in Table 7 for future comparison. The cutoff time for AWLS is 1 h, while Quintiq does not disclose the cutoff time limits.

In Table 7, columns "UB Reference" identifies the algorithm that established the new world records. The labels "[Q]," "[CPO]," and "[MAE]" denote the Quintiq method, CPO, and MAE, respectively. Finally, Table 8 presents a comprehensive comparison of AWLS (with a 1 h time limit) [39] against the meta-heuristic MAE, exact

Table 6
Comparison results with CPO on 46 instances.

Ins.	AWLS UB	CPO UB	CPO LB	Ins.	AWLS UB	CPO UB	CPO LB
Mk05	172	173	168	rdata-la22	753	755	741
Mk10	194	195	183	rdata-la23	830	832	816
02a	2228	2234	2228	rdata-la24	795	805	775
05a	2203	2213	2189	rdata-la25	779	787	768
06a	2185	2191	2162	rdata-la26	1057	1066	1056
07a	2249	2277	2216	rdata-la27	1085	1099	1085
08a	2062	2066	2061	rdata-la28	1076	1079	1075
10a	2241	2263	2197	rdata-la29	994	1001	993
11a	2044	2067	2017	rdata-la30	1070	1089	1068
12a	2009	2013	1969	rdata-la31	1520	1522	1520
13a	2232	2258	2197	rdata-la32	1657	1658	1657
14a	2162	2163	2161	rdata-la33	1497	1498	1497
15a	2161	2162	2161	rdata-la34	1535	1536	1535
16a	2226	2240	2148	vdata-car1	5005	5006	5005
17a	2121	2140	2088	vdata-car3	5597	5599	5597
18a	2112	2125	2057	vdata-car5	4910	4912	4909
edata-abz7	610	620	564	vdata-la22	733	734	733
edata-abz8	636	639	586	vdata-la25	752	753	751
rdata-abz7	522	535	492	vdata-la29	993	994	993
rdata-abz8	534	558	506	vdata-la30	1068	1069	1068
rdata-abz9	535	553	497	vdata-la32	1657	1658	1657
rdata-car3	5622	5623	5597	vdata-la33	1497	1498	1497
rdata-la21	824	838	808	vdata-la35	1549	1550	1549

Table 7
New world records obtained by AWLS.

Ins.	Previous world record					AWLS
	LB	LB Reference	UB	UB Reference	UB Date	UB
07a	2216	[CPO]	2254	[MAE]	Feb 2019	2249
13a	2197	[CPO]	2236	[MAE]	Feb 2019	2227
16a	2193	[CPO]	2231	[Q]	Jan 2016	2224
rdata-abz8	507	[Q]	535	[MAE]	Feb 2019	534
rdata-abz9	517	[CPO]	536	[Q]	Jan 2016	535
rdata-la21	808	[Q]	825	[Q]	Jan 2014	824
rdata-la23	816	[Q]	831	[MAE]	Feb 2019	830
rdata-la30	1068	[Q]	1071	[Q]	Jan 2016	1070

Table 8
Results on the all benchmark.

Set	AWLS vs MAE (1 h)			AWLS vs CPO (8 h)			AWLS vs Quintiq		
	<	=	>	<	=	>	<	=	>
DPdata	7	8	3	14	4	0	4	4	7
BCdata	2	19	0	0	21	0	0	0	0
BRdata	0	9	1	2	8	0	0	2	0
HUdata/edata	4	62	0	2	63	1	0	20	0
HUdata/rdata	6	60	0	18	48	0	9	27	1
HUdata/vdata	0	66	0	10	53	3	1	22	6
HUdata/sdata	3	63	0	0	66	0	0	18	0
Total	22	287	4	46	263	4	14	93	14

algorithm CPO, and industrial solver Quintiq. In this table, the columns < , = , and > indicate the count of instances where AWLS achieves better, equal, and worse results than others, respectively. It is noteworthy that AWLS matches all the world records obtained by MAE.

5. Discussion and analysis

To evaluate the merits of the adaptive weighting technique, we conducted experiments on the four most challenging instances in the BCdata benchmark. Several simplified variants of AWLS were generated by removing the impact of idle count when estimating makespan (WLS, $Z(i) = w_i$) and deactivating the adaptive weighting technique (LS, i.e., a standard TS algorithm), respectively. Furthermore, we conducted comparative experiments of our AWLS with the MAE algorithm, which is currently recognized as the best-performing meta-heuristic algorithm for FJSP. An extended version of MAE (MAE_AW) was also tested, where the LS procedure in MAE was replaced with LS using our adaptive weighting technique. All reference algorithms started from the same initial solution to ensure a fair comparison. Fig. 2 depicts the evolution trend of the makespan as the search proceeds by AWLS, WLS, LS, MAE_AW, and MAE. Each point (x, y) means that the makespan of the best-known solution at x iteration is y. The shaded area around the curve indicates the confidence interval of the data (average value ± standard deviation).

From Fig. 2, one observes that AWLS and WLS outperform LS by quickly obtaining better solutions. In contrast, WLS gets trapped in local optima and fails to obtain a better solution after several iterations. However, AWLS consistently enhances solution quality with the search progress. Regarding MAE and MAE_AW, MAE_AW demonstrates faster attainment of better solutions compared to MAE, highlighting the effectiveness of integrating adaptive weighted local search into diverse algorithms to achieve competitive results. The faster convergence of AWLS compared to MAE_AW may be attributed to the time-consuming nature of managing populations in MAE_AW, which is a population-based algorithm. These findings underscore that the idle count and adaptive weighting technique are both critical for our AWLS in terms of both effectiveness and efficiency.

Furthermore, we applied a statistical significance test (Wilcoxon signed-rank test) on the results of all benchmarks. Table 9 reports

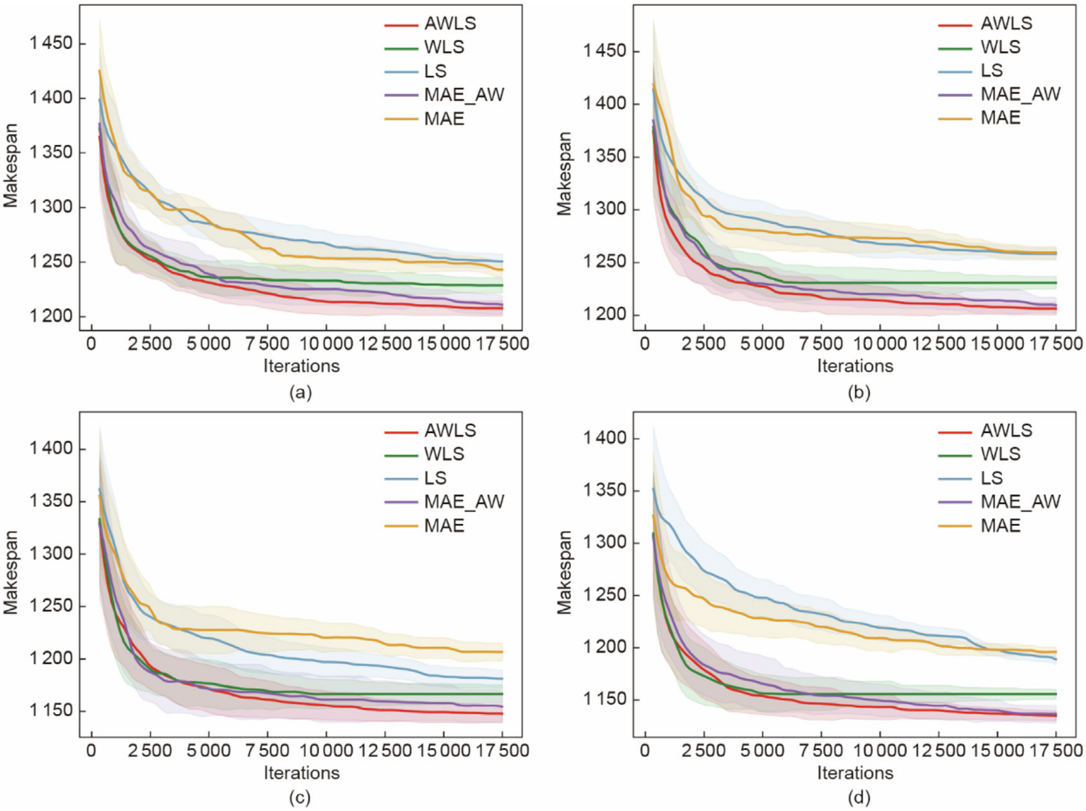

Fig. 2. Evolution of the makespan obtained by AWLS, its variants, and reference algorithms on four hardest instances in BCdata. (a) Seti5xx instance, (b) seti5xxx instance, (c) seti5xy instance, (d) seti5xyz instance

Table 9
Wilcoxon signed-rank test.

Set	AWLS vs SSPR	AWLS vs GRASP-m ELS	AWLS vs MAE
rdata	5.95×10^{-5}	5.38×10^{-5}	8.95×10^{-3}
edata	9.81×10^{-4}	5.06×10^{-3}	6.55×10^{-4}
DPdata	1.04×10^{-3}	8.05×10^{-4}	4.68×10^{-2}
BCdata	4.01×10^{-3}	1.25×10^{-2}	9.81×10^{-4}
vdata	2.21×10^{-3}	4.36×10^{-4}	0.8117
BRdata	0.2851	6.78×10^{-2}	0.2851

the resulting *p*-value on four benchmarks. With a significance level of 0.05, there are sharp differences between AWLS and the two reference algorithms, GRASP-mELS and SSPR, across all benchmarks except for BRdata. Moreover, sizeable differences are observed between AWLS and MAE on BCdata, DPdata, edata, and rdata, whereas no significant differences are found between AWLS and MAE on BRdata and vdata. The underlying reason might be that most instances in sets BRdata and vdata can be easily solved.

6. Conclusions

In this paper, we propose a novel adaptive weighting-based local search algorithm called AWLS to solve FJSP. The adaptive weighting technique assigns weights to each operation based on the idle counts and cumulative weight, treating these weights as additional adaptive processing time during makespan estimation, which can smooth the landscape of the search space. The experiment results conducted on 313 public benchmark instances show that AWLS outperforms most state-of-the-art algorithms. Moreover, AWLS updates the world records on 8 challenging instances. Thus, exploring the integration of the proposed strategies for solving other challenging scheduling problems would be an intriguing avenue for future research.

CRediT authorship contribution statement

Junjie Zhang: Data curation, Software, Writing – original draft. **Zhipeng Lü:** Funding acquisition, Supervision, Writing – review & editing. **Junwen Ding:** Software. **Zhouxing Su:** Software. **Xinyu Li:** Writing – review & editing. **Liang Gao:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (NSFC) (62202192 and 72101094) and the National Science Fund for Distinguished Young Scholars of China (51825502).

References

[1] Chen B, Wan J, Shu L, Li P, Mukherjee M, Yin B. Smart factory of industry 4.0: key technologies, application case, and challenges. IEEE Access 2018;6:6505–19.
[2] Wang B, Tao F, Fang X, Liu C, Liu Y, Freiheit T. Smart manufacturing and intelligent manufacturing: a comparative review. Engineering 2021;7(6):738–57.
[3] Li B, Hou B, Yu W, Lu X, Yang C. Applications of artificial intelligence in intelligent manufacturing: a review. Frontiers Inf Technol Electronic Eng 2017;18(1):86–96.
[4] Yang C, Liao F, Lan S, Wang L, Shen W, Hang G. Flexible resource scheduling for software-defined cloud manufacturing with edge computing. Engineering 2023;22:60–70.
[5] Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Math Oper Res 1976;1(2):97–196.
[6] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 1993;41(3):157–83.
[7] Chaudhry IA, Khan AA. A research survey: review of flexible job shop scheduling techniques. Int Trans Oper Res 2016;23(3):551–91.
[8] Ding J, Lü Z, Li C, Shen L, Xu L, Glover F. et al. A two-individual based evolutionary algorithm for the flexible job shop scheduling problem. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence; 2019 Jan 27–Feb 1; Honolulu, HI, USA. Washitong, DC: AIII Press; 2019. p. 2262–71.
[9] Zhang Y, Zhu H, Tang D, Zhou T, Gui Y. Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. Robot Comput-Integr Manuf 2022;78:102412.
[10] Du Y, Li J, Li C, Duan P. A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. IEEE Trans Neural Netw Learn Syst 2024;35(4):5695–709.
[11] Sawik T. Modelling and scheduling of a flexible manufacturing system. Eur J Oper Res 1990;45(2–3):177–90.
[12] Gomes MC, Barbosa-Póvoa AP, Novais AQ. Optimal scheduling for flexible job shop operation. Int J Prod Res 2005;43(11):2323–53.
[13] Elazeem A, Elazeem AE, Sayed M, Osman AF, Bayoumi M, Hassan A. Optimality of the flexible job shop scheduling problem. Afr J Math Comput Sci Res 2011;4(10):321–8.
[14] Gomes MC, Barbosa-Póvoa AP, Novais AQ. Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach. Int J Prod Res 2013;51(17):5120–41.
[15] Yin L, Li X, Gao L, Lu C, Zhang Z. A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem. Sustain Comput Inform Syst 2017;13:15–30.
[16] Kaplanoğlu V. An object-oriented approach for multi-objective flexible job-shop scheduling problem. Expert Syst Appl 2016;45:71–84.
[17] Liu Q, Li X, Gao L. Mathematical modeling and a hybrid evolutionary algorithm for process planning. J Intell Manuf 2021;32(3):781–97.
[18] Pezzella F, Morganti G, Ciaschetti G. A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 2008;35(10):3202–12.
[19] Gao J, Sun L, Gen M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 2008;35(9):2892–907.
[20] Zhang G, Shao X, Li P, Gao L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 2009;56(4):1309–18.
[21] Zhou G, Wang L, Xu Y, Wang S. An effective artificial bee colony algorithm for multi-objective flexible job-shop scheduling problem. In: Huang DS, Gan Y, Gupta P, Gromiha MM, editors. Advanced Intelligent Computing Theories and Applications; 2011 Aug 11–14; Zhengzhou, China. Berlin: Springer; 2011. p. 1–8.
[22] González MA, Vela CR, Varela R. Scatter search with path relinking for the flexible job shop scheduling problem. Eur J Oper Res 2015;245(1):35–45.
[23] Palacios JJ, González MA, Vela CR, González-Rodríguez I, Puente J. Genetic tabu search for the fuzzy flexible job shop problem. Comput Oper Res 2015;54:74–89.
[24] Li X, Gao L. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. Int J Prod Econ 2016;174:93–110.
[25] Kemmoé-Tchomté S, Lamy D, Tchernev N. An effective multi-start multi-level evolutionary local search for the flexible job-shop problem. Eng Appl Artif Intell 2017;62:80–95.
[26] Caldeira RH, Gnanavelbabu A. Solving the flexible job shop scheduling problem using an improved Jaya algorithm. Comput Ind Eng 2019;137:106064.
[27] Fan J, Shen W, Gao L, Zhang C, Zhang Z. A hybrid Jaya algorithm for solving flexible job shop scheduling problem considering multiple critical paths. J Manuf Syst 2021;60:298–311.
[28] Li R, Gong W, Wang L, Lu C, Jiang S. Two-stage knowledge-driven evolutionary algorithm for distributed green flexible job shop scheduling with type-2 fuzzy processing time. Swarm Evol Comput 2022;74:101139.
[29] Xie J, Li X, Gao L, Gui L. A hybrid genetic tabu search algorithm for distributed flexible job shop scheduling problems. J Manuf Syst 2023;71:82–94.
[30] Sun K, Zheng D, Song H, Cheng Z, Lang X, Yuan W, et al. Hybrid genetic algorithm with variable neighborhood search for flexible job shop scheduling problem in a machining system. Expert Syst Appl 2023;215:119359.
[31] Yang D, Wu M, Li D, Xu Y, Zhou X, Yang Z. Dynamic opposite learning enhanced dragonfly algorithm for solving large-scale flexible job shop scheduling problem. Knowl-Based Syst 2022;238:107815.
[32] Alzaqebah M, Jawarneh S, Alwohaibi M, Alsmadi MK, Almarashdeh I, Mohammad RMA. Hybrid brain storm optimization algorithm and late acceptance hill climbing to solve the flexible job-shop scheduling problem. J King Saud University-Comput Inf Sci 2022;34(6):2926–37.
[33] Mastrolilli M, Gambardella LM. Effective neighbourhood functions for the flexible job shop problem. J Sched 2000;3(1):3–20.
[34] Peng B, Lü Z, Cheng TCE. A tabu search/path relinking algorithm to solve the job shop scheduling problem. Comput Oper Res 2015;53:154–64.

[35] Dauzère-Pérès S, Paulli J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 1997;70:281–306.

[36] Brandimarte P. Routing and scheduling in a flexible job shop by Tabu search. Ann Oper Res 1993;41:157–83.

[37] Hurink J, Jurisch B, Thole M. Tabu search for the job-shop scheduling problem with multi-purpose machines. Oper Res Spektrum 1994;15(4):205–15.

[38] Vilím P, Laborie P, Shaw P. Failure-directed search for constraint-based scheduling. In: Michel L, editor. Integration of AI and OR Techniques in Constraint Programming; 2015 May 18–22; Barcelona, Spain. Berlin: Springer; 2015. p. 437–53.

[39] Zhang J. The solution files and their corresponding Gantt charts obtained by AWLS [Internet]. 2023 [2024 Jun 28]. Available from: https://github.com/Zoommy/FJSP_AWLS.